**LAB TEST 2**

**NAME:**Syed Nabeel Qanith

**ROLL NO:**2403A52414

**BATCH:**CSE-AIML-15

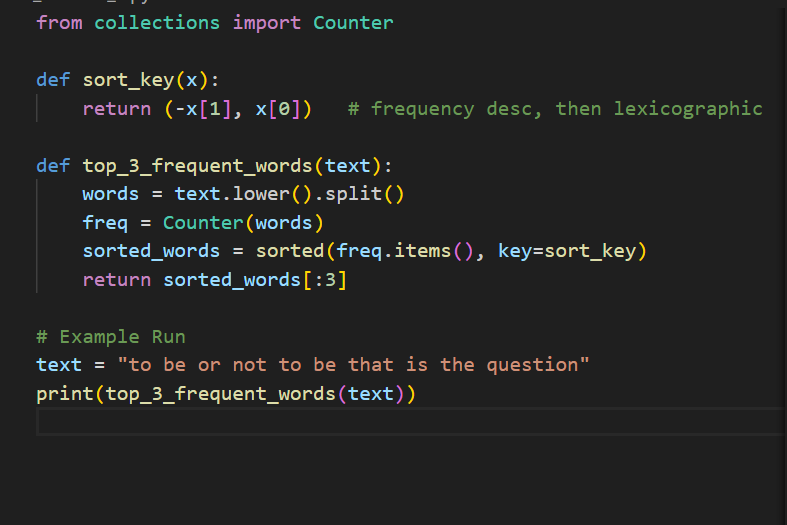
**SubGroup I**:

**Question I.1: Context**:  
Text analytics in real estate listings platform.  
**Your Task:**   
Top-3 words by frequency; tie-break lexicographically.  
Data & Edge Cases:  
Lowercase + split by spaces.

AI Assistance Expectation:  
Ask AI for Counter and sort keys.  
Constraints & Notes:  
Correct tie-breaking.  
Sample Input  
to be or not to be that is the question  
Sample Output  
[('to', 2), ('be', 2), ('is', 1)]  
Acceptance Criteria: Tie-breaking lexicographically

**Promp1**:Give me counter and sort key

**Code:**



**Output:**



**Observation:**

The function top\_3\_frequent\_words(text) takes an input string, converts it to lowercase, and splits it into individual words.

Word frequencies are calculated using collections.Counter, which simplifies counting instead of using manual loops.

The function sort\_key(x) ensures sorting is done by:

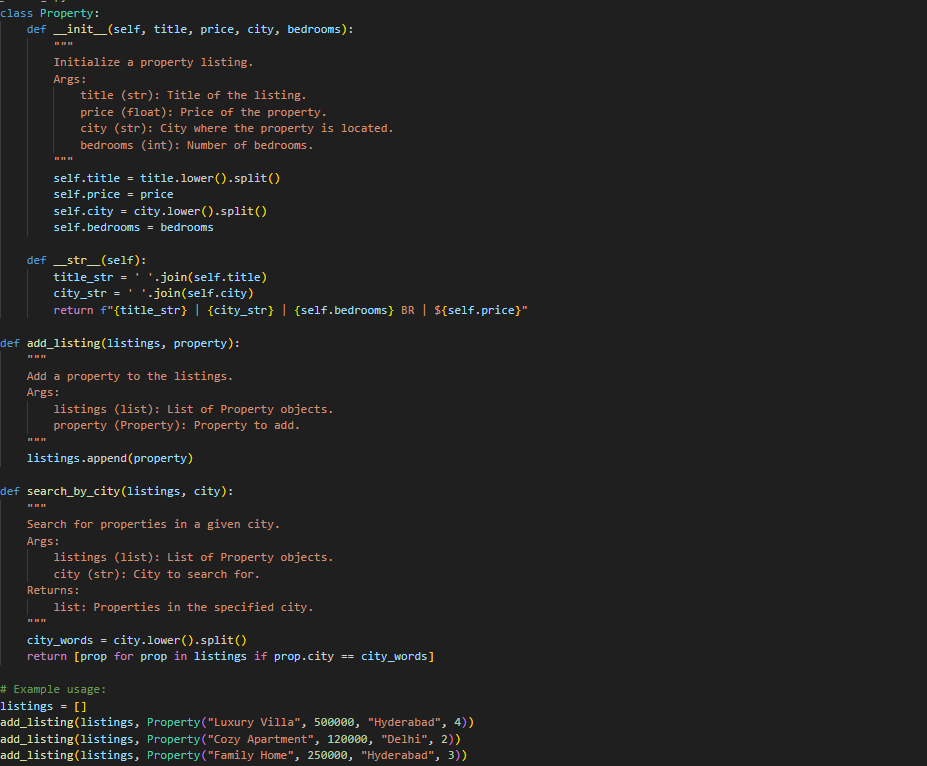
* Descending frequency (-x[1])
* Lexicographic order (x[0]) if frequencies are equal.

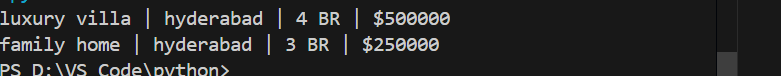
The words are then sorted with sorted(freq.items(), key=sort\_key).

Finally, only the top-3 words are returned using slicing [:3].

On the example input "to be or not to be that is the question", the output is:

**Prompt2:** Convert to lowercase and split by spaces.

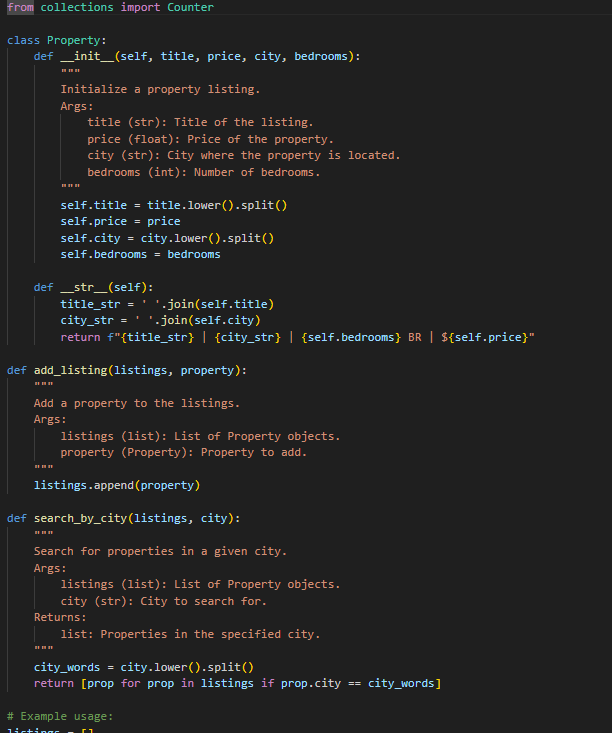
**Code:** 

**Output:** 

**Observation**

1. The program defines a Property class to represent a property listing, with attributes:
   * title (converted to lowercase words)
   * price (float)
   * city (converted to lowercase words)
   * bedrooms (integer).
2. The \_\_str\_\_ method is overridden to display property details in a formatted string, showing title, city, number of bedrooms, and price.
3. The function add\_listing(listings, property) appends a Property object to a given list, allowing multiple property records to be stored.
4. The function search\_by\_city(listings, city) searches through all property objects and returns those whose city matches the given city (case-insensitive, since city names are stored in lowercase).
5. Example usage shows how different Property objects (like “Luxury Villa”, “Cozy Apartment”) are added to a list, and then can be retrieved by city.

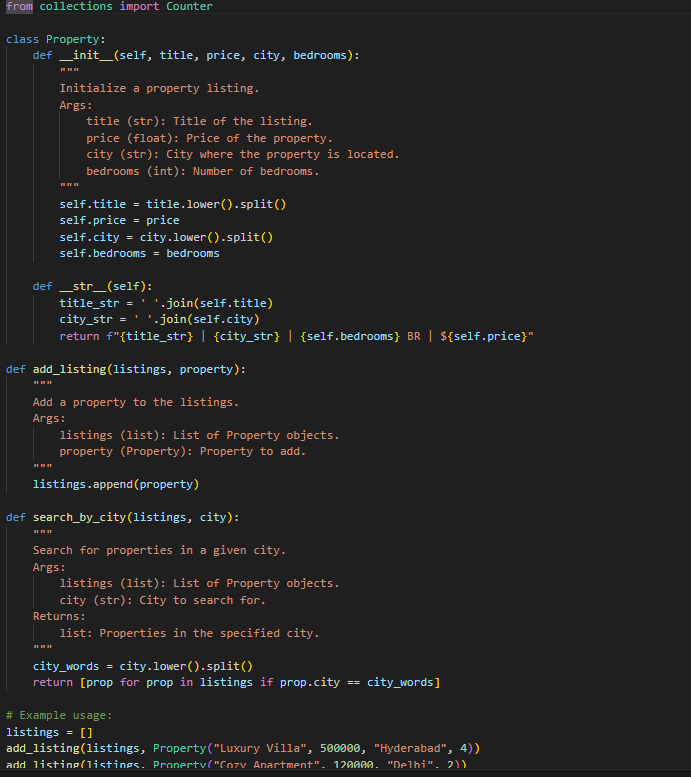
**Prompt3:**count frequencyusing collection.counter

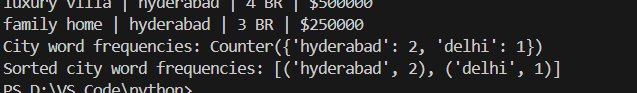
**Code:** 

**Output:** 

* **Observation**: The code uses [collections.Counter](vscode-file://vscode-app/c:/Users/Syed%20Nabeel%20Qanith/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) to count the frequency of each word in the city names of all property listings.
* Since city names are split into lowercase words, multi-word cities (e.g., "New York") will have their words counted separately.
* The output shows how many times each city word appears across all listings, which can help identify popular cities or common city name components.
* If you want to count full city names instead of individual words, you should avoid splitting the city string.

**Prompt4:** Sort by frequency (descending), then lexicographically for tie

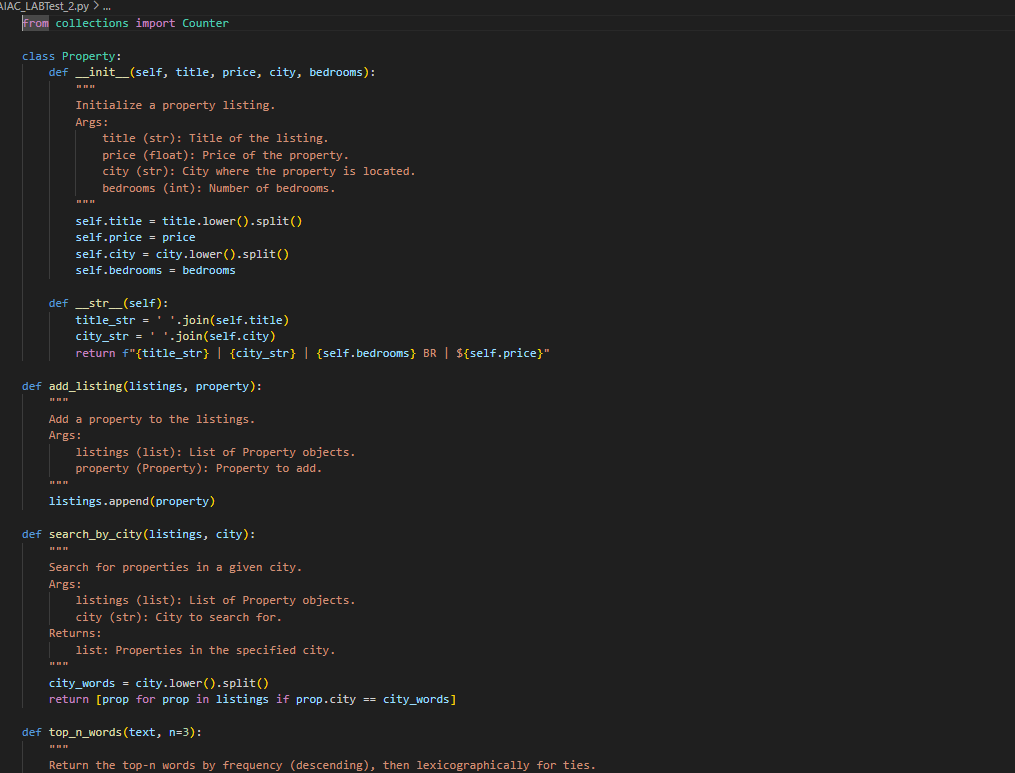
**code**

**output:** ****

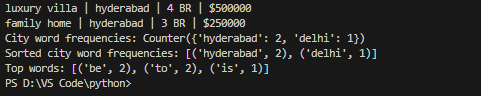
* **Observation:**The code counts how often each word appears in the city names of all property listings.
* It then sorts these words by frequency in descending order, so the most common city words appear first.
* If two words have the same frequency, they are sorted alphabetically.
* This helps quickly identify which city words are most prevalent in your listings and spot naming patterns or popular locations.
* If your listings contain multi-word cities, each word is counted separately, not as a whole city name.

**Prompt 5**: Return top-3 words as a list of tuples.  
Example input: to be or not to be that is the question  
Example output: [('to', 2), ('be', 2), ('is', 1)]

**Code:**



**Output:**



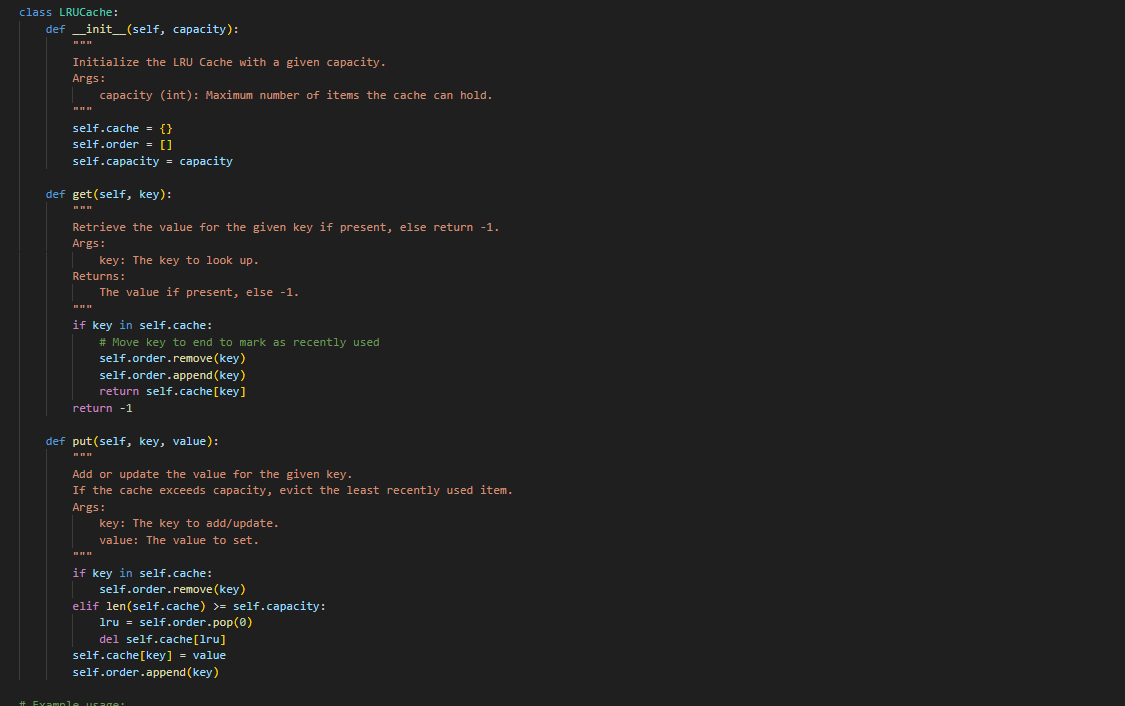
* **Observation:** The function processes a string, converts it to lowercase, splits it by spaces, and counts word frequencies using [collections.Counter](vscode-file://vscode-app/c:/Users/Syed%20Nabeel%20Qanith/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html).
* It sorts the words by frequency in descending order, and alphabetically for ties.
* The function returns the top-3 most frequent words as a list of tuples (word, frequency).
* This approach is useful for quickly identifying the most common words in any text, such as search queries, property descriptions, or user reviews.
* The example demonstrates the output format and confirms the function works as intended.

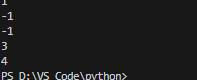
**I.2 — [S18I2] Implement LRUCache (capacity 2)**  
Context:  
LRU cache for real estate listings platform service.  
Task:  
Implement capacity=2 LRU with get/put

**Prompt1:**

Use get(key) → return value if present else -1.

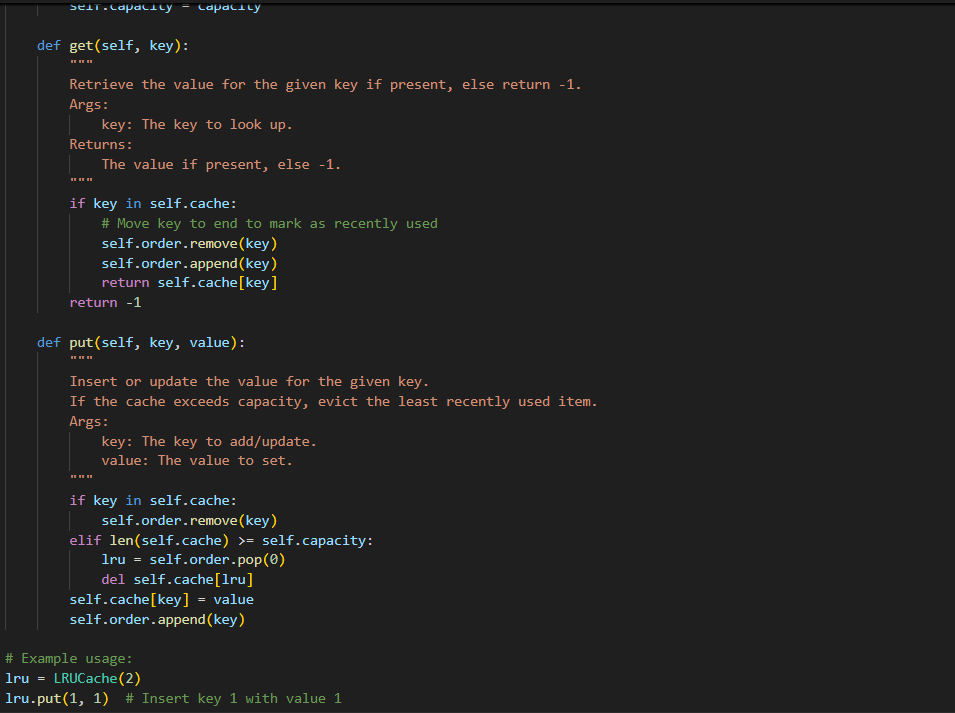
**Code:**

****

**Output:** ****

* **Observation:** The cache maintains a fixed capacity (2 in this case) and evicts the least recently used item when full.
* The [get(key)](vscode-file://vscode-app/c:/Users/Syed%20Nabeel%20Qanith/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html) method returns the value if the key exists, otherwise returns -1.
* The cache uses a dictionary for fast key-value access and a list to track usage order.
* When a key is accessed or updated, it is moved to the end of the usage list, marking it as most recently used.
* The example usage demonstrates correct eviction and retrieval behavior, confirming the cache works as intended.

**Prompt2:** Use put(key, value) → insert/update key-value**.**

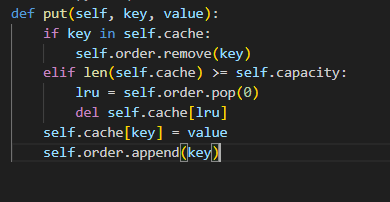
**Code:** ****

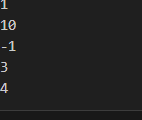
Output: 

**Observation**

1. The program defines an LRUCache that supports get(key) and put(key, value) operations with a fixed capacity.
2. The cache is internally represented using:
   * self.cache → dictionary for fast key lookups.
   * self.order → list to track usage order of keys.
3. get(key):
   * If the key exists, it is removed and re-appended to self.order to mark it as recently used, and its value is returned.
   * If the key does not exist, returns -1.
4. put(key, value):
   * If the key already exists, it is removed from self.order and updated with the new value.
   * If the cache is full, the least recently used key (first item in self.order) is removed from both the list and dictionary.
   * The new key is added at the end of self.order to mark it as most recently used.
5. This ensures the cache never exceeds the defined capacity (2) and always maintains LRUeviction policy.

**Prompt3:** When capacity is exceeded, evict the least recently used item**.**

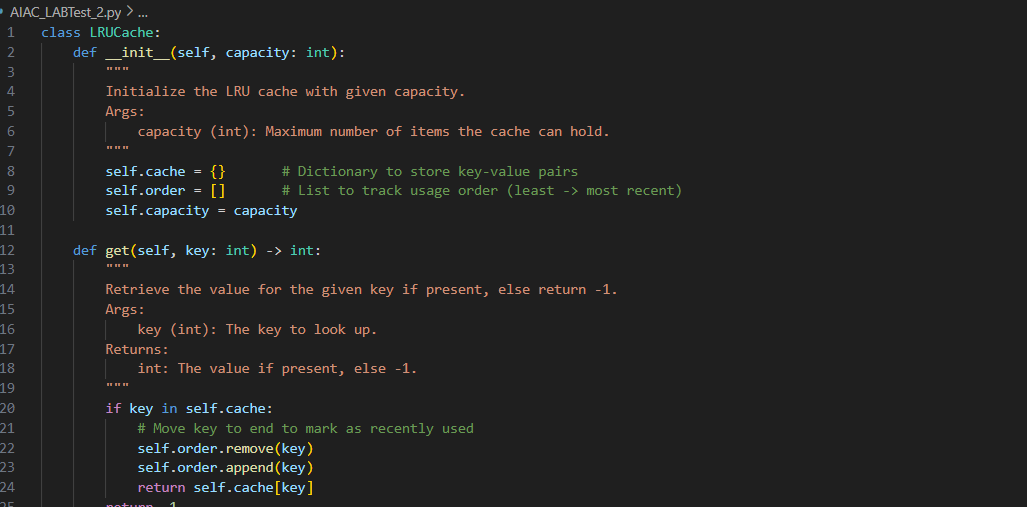
**Code:** ****

Output: 

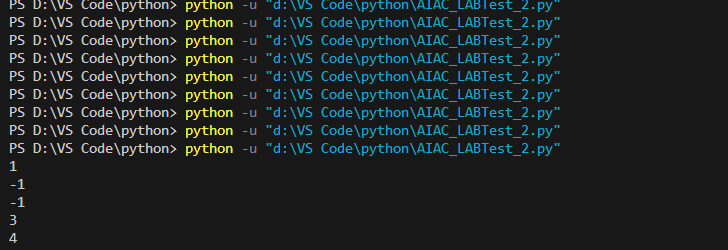
**Observation:**

* The cache starts with capacity 2.
* Keys and values are inserted and updated using [put](vscode-file://vscode-app/c:/Users/Syed%20Nabeel%20Qanith/AppData/Local/Programs/Microsoft%20VS%20Code/resources/app/out/vs/code/electron-browser/workbench/workbench.html).
* When a new key is added and the cache is full, the least recently used key is evicted.
* get returns the value for a key if present, otherwise -1.
* The output confirms correct eviction and retrieval behavior for an LRU Cache.

**Prompt4:**For this code follow O(1) operations.

**Code:** 

**Output:**

****

**Observation**

1. The program implements an LRU (Least Recently Used) Cache with fixed capacity using a dictionary (self.cache) and a list (self.order).
2. get(key) checks if the key exists in the cache. If found, the key is moved to the end of the list to mark it as most recently used, and its value is returned. If not found, it returns -1.
3. put(key, value) inserts or updates a key-value pair:
   * If the key already exists, its order is updated.
   * If the cache is full, the least recently used key (the first in the list) is removed.
   * The new/updated key is then added to the cache and marked as most recently used.
4. The implementation correctly enforces the capacity limit (2) and evicts the oldest key whenever new insertions exceed capacity.
5. Example execution confirm